Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение  
высшего образования «Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»

Институт Информационных технологий, математики и механики

Отчёт по лабораторной работе

Таблицы

Выполнил:

студент гр. 381603-01

Жирнов В.Г

Проверил:

ассистент каф. МОСТ, ИИТММ

Кустикова В.Д.

Нижний Новгород

2018 г

Содержание

[Введение 3](#_Toc515842397)

[1 Постановка задачи 4](#_Toc515842398)

[2 Руководство пользователя 5](#_Toc515842399)

[3 Руководство программиста 8](#_Toc515842400)

[3.1 Структура программы 8](#_Toc515842401)

[3.2 Структуры данных 9](#_Toc515842402)

[3.2.1 Шаблонный класс TabRec 9](#_Toc515842403)

[3.2.2 Виртуальный шаблонный класс Table 9](#_Toc515842404)

[3.2.3 Дочерний шаблонный класс ScanTab 10](#_Toc515842405)

[3.2.4 Дочерний шаблонный класс SortTab 10](#_Toc515842406)

[3.2.5 Дочерний шаблонный класс HashTab 10](#_Toc515842407)

[3.3 Описание алгоритмов 11](#_Toc515842408)

[3.3.1 Несортируемые таблицы 11](#_Toc515842409)

[3.3.2 Сортируемые таблицы 11](#_Toc515842410)

[3.3.3 Хеш таблицы 12](#_Toc515842411)

[Заключение 14](#_Toc515842412)

[Литература 15](#_Toc515842413)

[Приложение 16](#_Toc515842414)

# Введение

Таблица — набор элементов одинаковой организации, каждый из которых можно представить в виде двойки <K, V>, где K — ключ, а V — тело (информационная часть) элемента.

Ключ уникален для каждого элемента, то есть в таблице нет элементов с одинаковыми ключами. Ключ используется для доступа к элементам при выполнении операций.

Виды таблиц:

Существует три вида таблиц: неупорядоченная, упорядоченная и хеш-таблица. Рассмотрим подробнее каждую из них.

Элементы такой таблицы не упорядочены по значению ключа. Для поиска элемента с заданным ключом используется алгоритм линейного или быстрого линейного поиска

В такой таблице все элементы упорядочиваются по возрастанию значения ключа. Для поиска элемента с заданным ключом применяются алгоритмы линейного, бинарного или блочного поиска. После выполнения операции включения, элементы таблицы должны остаться упорядоченными, поэтому перед выполнением операции включения необходимо найти элемент, после которого нужно включить элемент.

Хеш-таблица — это таблица, в которой положение адреса элемента определяется с помощью некоторой функции H (хеш-функции), аргументом которой является значение ключа элемента.

# Постановка задачи

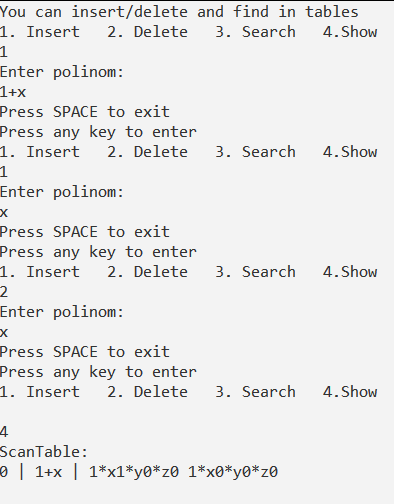
Реализовать шаблонный виртуальный абстрактный класс таблиц, который содержит в себе 3 основные операции: вставка, удаление, поиск, а также методы навигации. На основе этого класса реализовать три дочерних: классы сортируемых таблиц, несортируемых таблиц и хеш-таблиц. Написать основную программу, позволяющую пользователю в консоли работать с тремя видами таблиц: вставлять, удалять и искать многочлены, а также просматривать таблицы.

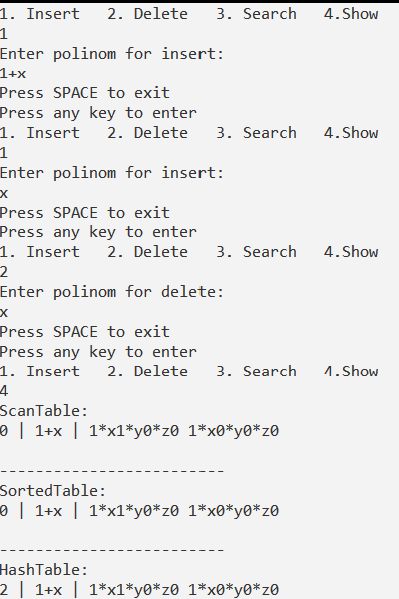
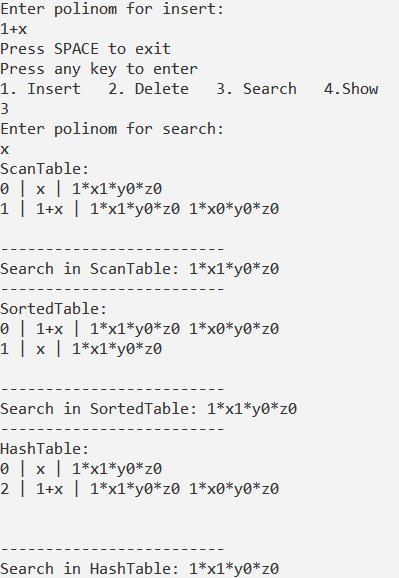
Исходные данные: 3 таблицы.

Выходные данные: Состояние таблицы, после взаимодействия с пользователем (вставки, удаления многочлена).

# Руководство пользователя

1. Открыть файл Debug\sample.exe
2. Появится диалоговое окно -> нажать на цифру от 1-4
3. Ввести многочлен при нажатии 1, 2 или 3
4. После выполнения операции нажать пробел, чтобы выйти из программы или любую другую клавишу, чтобы продолжить работу



1. Демонстрация вставки многочлена
2. Демонстрация удаления многочлена
3. Демонстрация поиска многочлена

# Руководство программиста

## Структура программы

1. В программе содержится статическая библиотека polinom\_lib.lib, описанная в отчете «Операции над полиномами»
2. Table.h Содержит объявление класса записей(TabRec) и абстрактного класса таблиц(Table). Так же, здесь реализованы методы навигации, выделение памяти и прочие методы виртуального шаблонного класса Table, а так же методы класса TabRec.
3. Sort\_table.h Содержит объявление дочернего класса SortTab – сортируемых таблиц, унаследованного от класса Table. Так же, здесь реализованы методы вставки, удаления, бинарного поиска и поиска элемента таблицы-записи.
4. Scan\_table.h h Содержит объявление дочернего класса ScanTab – несортируемых таблиц, унаследованного от класса Table. Так же, здесь реализованы методы вставки, удаления, поиска элемента таблицы-записи
5. Hash\_table.h h Содержит объявление дочернего класса HashTab – hash-таблиц, унаследованного от класса Table. Так же, здесь реализованы методы вставки, удаления, поиска элемента таблицы-записи, методы навигации, и хеш-функция.
6. Sample.cpp Содержит реализацию пользовательского приложения
7. Test\_sort.cpp Содержит тесты для проверки правильной работоспособности сортируемых таблиц
8. Test\_scan.cpp Содержит тесты для проверки правильной работоспособности несортируемых таблиц
9. Test\_hash.cpp Содержит тесты для проверки правильной работоспособности сортируемых таблиц

## Структуры данных

### Шаблонный класс TabRec

Поля(public):

string key; ключ

val datа; данные

Методы(public):

TabRec()конструктор по умолчанию

TabRec(const string& s, const val& d) конструктор

TabRec(const TabRec& t) конструктор копирования

TabRec& operator= (const TabRec<val>& t) перегрузка оператора присваивания

### Виртуальный шаблонный класс Table

Поля(protected):

TabRec<val>\*\* linerec; Массив записей(представление таблицы)

int maxrec; максимальное количество записей

int currrec; текущее количество записей

int currindex; текущий индекс

virtual void Realloc(); выделение памяти

Методы(public):

Table(int i = 10); конструктор по умолчанию

Table(const Table<val>& t); конструктор копирования

virtual ~Table() { delete[] linerec; } деструктор

virtual void insert(const string& tempkey, const val& tempdata) = 0; чистый виртуальный метод – вставка записи

virtual void del(const string &k) = 0; чистый виртуальный метод – удаление записи

virtual val& search(const string& tempkey) const = 0; чистый виртуальный метод – поиск записи

virtual void reset(); возврат в начало таблицы

virtual bool IsTabEnd() const { return ((currindex+1) == currrec) || currindex == -1; } проверка на конец таблицы

bool IsFull() const { return currrec == maxrec; } проверка на полноту таблицы

bool IsEmpty() const { return currrec == 0; } проверка на пустоту таблицы

virtual void set(); переход на следующую запись

virtual val& getcurr() const; получение данных из текущей записи

virtual int getcurrind() { return currindex; } получение текущего индекса

virtual int getcurrrec() { return currrec; } количество текущих записей

template <class val> friend ostream& operator<< (ostream& os, const Table<val>& t); вывод таблицы на экран

### Дочерний шаблонный класс ScanTab

Все методы и поля унаследованы из класса Table

### Дочерний шаблонный класс SortTab

Все методы и поля унаследованы из класса Table, плюс

int binsearch(string& key) const; бинарный поиск

### Дочерний шаблонный класс HashTab

Все методы и поля унаследованы из класса Table, плюс

Int\* flag; поле(public), флаг для проверки занятости записи

Int HashFunc (const string& key) const; хеш функция

## **Описание алгоритмов**

### Несортируемые таблицы

#### Вставка

* Проверка на полноту, если true, выделить больше памяти
* Если текущее количество записей равно нулю, то выводим ошибку
* Если больше, то перебирая записи(увеличивая текущий индекс) ищем запись с похожим ключом, если находим, то выводим ошибку, иначе доходим до конца таблицы и вставляем элемент в конец

#### Удаление

* Идет поиск нужной записи, если таковой нет, то выводится ошибка
* Если нашлась, то она удаляется и идет перепаковка записей

#### Поиск

* Идет поиск нужной записи, если таковой нет, то выводится ошибка
* Если нашлась, то из нее извлекаются и выводятся данные на экран

### Сортируемые таблицы

#### Вставка

* Проверка на полноту, если true, то выделяется больше памяти
* Если текущее количество записей меньше нуля выдается ошибка
* Если текущее количество записей больше нуля, то с помощью бинарного поиска находим индекс входной записи, где она должна находиться в упорядоченной таблице
* Перемещаем все записи, ключ которых больше входной записи на 1 вниз
* Вставляем входную запись на освободившееся место

#### Удаление

* С помощью бинарного поиска находится запись и удаляется если она существует. Если нет, то выводится ошибка
* Перемещаем все элементы, ключ которых больше ключа удаленной записи, на 1 вверх

#### Поиск

* С помощью бинарного поиска находится индекс нужной записи, затем её данные извлекаются и выводятся
* Если такой записи не существует, выводится ошибка

#### Бинарный поиск

* Определяется значение ключа в середине массива указателей
* Ключ сравнивается с этим значением и если он больше, то поиск продолжается во второй половине элементов, еслиХеш меньше – в первой
* Далее ищется половина от предыдущей половины и ключ сравнивается уже с этим значением
* Данные действия происходят и дальше, пока ключ не будет найден, или интервал для поиска не будет пустым

### Хеш таблицы

#### Хеш функция

* Каждый символ ключа переводится в число и прибавляется к переменной hash
* Вывод результата деления с остатком hash на максимальное количество записей

#### Вставка

* Идет проверка на полноту таблицы, если полна, то идет увеличение памяти
* С помощью хеш функции для входного ключа получаем индекс
* Если нет коллизии(когда запись с этим индексом уже занята) то запись создается в этой ячейке, если коллизия присутствует, то запись создается в следующей свободной ячейке таблицы
* Если в таблице уже есть запись с таким ключом, то выводится ошибка

#### Удаление

* С помощью хеш функции для входного ключа получаем индекс
* Если нет коллизии, то запись удаляется и флажку присваивается 0 для этого индекса, если она есть то идем по записям дальше и ищем запись ключ которой совпадает с входным ключом.
* Если таких записей нет или таблица пустая, выводится ошибка

#### Поиск

* С помощью хеш функции для входного ключа получаем индекс
* Если нет коллизии, то выводятся данные, лежащие в этом индексе; если она есть то идем по записям дальше и ищем запись ключ которой совпадает с входным ключом.
* Если таких записей нет или таблица пустая, выводится ошибка

# Заключение

В данной лабораторный работе был реализован виртуальный шаблонный класс таблиц, поддерживающий операции вставки, удаления и поиска, а также реализованы на основе этого класса 3 типа таблиц: хеш таблицы, сортируемые и несортируемые. Реализована программа – интерфейс, предоставляющая пользователю возможность вводить, удалять, искать полиномы в данных таблицах.

# Литература

1. Лекции «Алгоритмы и структуры данных» Гергель В.П. ННГУ им. Лобачевского Н. И.

# Приложение

##### Программная реализация шаблонного класса TabRec и виртуального шаблонного класса Table

#pragma once

#include <string>

#include <iostream>

using namespace std;

template <typename val>

class TabRec

{

public:

string key;

val data;

TabRec() { key = "null"; }

TabRec(const string& s, const val& d) { key = s; data = d; }

TabRec(const TabRec& t) { key = t.key; data = t.data; }

TabRec& operator= (const TabRec<val>& t)

{

data = t.data;

key = t.key;

}

};

template <typename val>

class Table

{

protected:

TabRec<val>\*\* linerec;

int maxrec;

int currrec;

int currindex;

virtual void Realloc();

public:

Table(int i = 10);

Table(const Table<val>& t);

virtual ~Table() { delete[] linerec; }

virtual void insert(const string& tempkey, const val& tempdata) = 0;

virtual void del(const string &k) = 0;

virtual val& search(const string& tempkey) const = 0;

virtual void reset();

virtual bool IsTabEnd() const { return ((currindex+1) == currrec) || currindex == -1; }

bool IsFull() const { return currrec == maxrec; }

bool IsEmpty() const { return currrec == 0; }

virtual void set();

virtual val& getcurr() const;

virtual int getcurrind() { return currindex; }

virtual int getcurrrec() { return currrec; }

template <class val> friend ostream& operator<< (ostream& os, const Table<val>& t);

};

template<typename val>

Table<val>::Table(int i)

{

maxrec = i;

currrec = 0;

currindex = -1;

linerec = new TabRec <val>\*[maxrec];

}

template<typename val>

Table<val>::Table(const Table<val>& t)

{

currrec = t.currrec;

maxrec = t.maxrec;

currindex = t.currindex;

delete [] linerec;

linerec = new TabRec<val>\*[maxrec];

for (int i = 0; i < currrec; i++)

linerec[i] = t.linerec[i];

}

template <typename val>

void Table<val>::Realloc()

{

int newmax = maxrec \* 2;

TabRec<val>\*\* k = new TabRec<val>\*[newmax];

reset();

for (int i = 0; i < maxrec; i++)

k[i] = linerec[i];

delete[] linerec;

maxrec = newmax;

linerec = k;

}

template<typename val>

void Table<val>::reset()

{

if (currrec)

currindex = 0;

else

currindex = -1;

}

template<typename val>

val& Table<val>::getcurr() const

{

if (currrec)

return linerec[currindex]->data;

else

throw "table is empty";

}

template<typename val>

void Table<val>::set()

{

if (currrec)

currindex++;

else

throw "table is empty";

if (currindex==currrec)

reset();

}

template <typename val>

ostream& operator<< (ostream& os, const Table<val>& t)

{

if (t.currrec)

for (int i = 0; i < t.currrec; i++)

os << i << " | " << t.linerec[i]->key << " | " << t.linerec[i]->data << endl;

else

os << "Table is empty" << endl;

return os;

}

##### Программная реализация дочернего шаблонного класса ScanTab

#pragma once

#include "table.h"

template<typename val>

class ScanTab : public Table<val>

{

public:

ScanTab(int i = 10) : Table(i) {}

ScanTab(const ScanTab<val>& t) : Table(t) {}

~ScanTab() {}

void insert(const string& tempkey, const val& tempdata) override;

val& search(const string& key) const override;

void del(const string& key) override;

};

template<typename val>

void ScanTab<val>::insert(const string& tempkey, const val& tempdata)

{

if (IsFull())

Realloc();

reset();

if (currrec)

{

while ((currindex < currrec) && (linerec[currindex]->key != tempkey))

currindex++;

if (currindex==currrec)

{

linerec[currindex] = new TabRec<val>(tempkey,tempdata);

currrec++;

}

else

throw "this key is already exists";

}

else

{

currindex++;

linerec[currindex] = new TabRec<val>(tempkey,tempdata);

currrec++;

}

}

template<typename val>

val& ScanTab<val>::search(const string& tempkey) const

{

ScanTab<val> k(\*this);

k.reset();

if (k.currindex > -1)

{

while ((k.linerec[k.currindex]->key != tempkey) && (k.currindex < k.currrec))

k.set();

if ((k.currindex < k.currrec) && (k.linerec[k.currindex]->key == tempkey))

return k.linerec[k.currindex]->data;

else

throw "key is not found";

}

else

throw "table is empty";

}

template<typename val>

void ScanTab<val>::del(const string& key)

{

reset();

if (currindex > -1)

{

while ((currindex < currrec) && (linerec[currindex]->key != key))

currindex++;

if (currrec && (currindex < currrec))

{

if (currrec > 1)

linerec[currindex] = linerec[--currrec];

else

currrec = 0;

}

else

throw "key is not found";

}

else

throw "table is empty";

}

##### Программная реализация дочернего шаблонного класса SortTab

#pragma once

#include "table.h"

template<typename val>

class SortTab : public Table<val>

{

protected:

int binsearch(const string &key) const;

public:

SortTab(int i = 10) : Table(i) {}

SortTab(const SortTab<val>& t) : Table(t) {}

~SortTab() {}

void insert(const string& tempkey, const val& tempdata) override;

val& search(const string& tempkey) const override;

void del(const string& key) override;

};

template<typename val>

int SortTab<val>::binsearch(const string& tempkey) const

{

int i = 0, mid;

int j = currrec - 1;

while (i <= j)

{

mid = (i + j) / 2;

if (tempkey > linerec[mid]->key)

i = mid + 1;

else

j = mid - 1;

}

return i;

}

template<typename val>

void SortTab<val>::insert(const string& tempkey, const val& tempdata)

{

if (IsFull())

Realloc();

reset();

if (currrec)

{

int k = binsearch(tempkey);

if (k != currrec)

{

if (linerec[k]->key != tempkey)

{

for (int i = currrec; i > k; i--)

linerec[i] = linerec[i - 1];

linerec[k] = new TabRec<val>(tempkey, tempdata);

currrec++;

}

else

throw "key already exists";

}

else

{

linerec[k] = new TabRec<val>(tempkey, tempdata);

currrec++;

}

}

else

{

currindex++;

linerec[currindex] = new TabRec<val>(tempkey, tempdata);

currrec++;

}

}

template<typename val>

val& SortTab<val>::search(const string& tempkey) const

{

SortTab<val> temp(\*this);

temp.reset();

if (currindex > -1)

{

int k = binsearch(tempkey);

if (temp.linerec[k]->key == tempkey)

return temp.linerec[k]->data;

else

throw "key is not found";

}

else

throw "table is empty";

}

template<typename val>

void SortTab<val>::del(const string& key)

{

reset();

if (currrec)

{

int k = binsearch(key);

if ((k < currrec) && (linerec[k]->key == key))

{

for (int i = k; i < currrec; i++)

linerec[i] = linerec[i + 1];

currrec--;

}

else

throw "key is not found";

}

else

throw "table is empty";

}

##### Программная реализация дочернего шаблонного класса HashTab

#pragma once

#include "table.h"

template <typename val>

class HashTab : public Table<val>

{

protected:

int\* flag;

int HashFunc(const string& key) const;

void Realloc() override;

public:

HashTab(int i = 10);

HashTab(const HashTab<val>& t);

~HashTab() { delete[] flag; }

void insert(const string& tempkey, const val& tempdata) override;

val& search(const string& tepmkey) const override;

void del(const string& k) override;

void set() override;

val& getcurr() const override;

void reset() override;

bool IsTabEnd() const { return currindex == currrec || currindex == -1; }

template <class val> friend ostream& operator<<(ostream& out, const HashTab<val>& t);

};

template<typename val>

HashTab<val>::HashTab(int i = 5) : Table(i)

{

flag = new int[i];

for (int j = 0; j < maxrec; j++)

flag[j] = 0;

}

template<typename val>

int HashTab<val>::HashFunc(const string& key) const

{

int hash = 0;

for (int i = 0; i < key.length(); i++)

hash = hash + int(key[i]);

return hash % maxrec;

}

template<typename val>

void HashTab<val>::Realloc()

{

int newmax = maxrec \* 2;

TabRec<val>\*\* k = new TabRec<val>\*[newmax];

int\* tempflag = new int[newmax];

for (int i = 0; i < maxrec; i++)

{

k[i] = linerec[i];

tempflag[i] = flag[i];

}

delete[] linerec;

delete[] flag;

linerec = k;

flag = tempflag;

maxrec = newmax;

}

template<typename val>

HashTab<val>::HashTab(const HashTab<val>& t)

{

currrec = t.currrec;

maxrec = t.maxrec;

currindex = t.currindex;

delete[] linerec;

delete[] flag;

linerec = new TabRec<val>\*[maxrec];

flag = new int[maxrec];

for (int i = 0; i < maxrec; i++)

{

flag[i] = t.flag[i];

linerec[i] = t.linerec[i];

}

}

template<typename val>

void HashTab<val>::reset()

{

if (currrec)

{

currindex = 0;

while (flag[currindex] != 1)

currindex++;

}

else

currindex = -1;

}

template<typename val>

val& HashTab<val>::search(const string& tempkey) const

{

HashTab<val> temp(\*this);

temp.reset();

if (temp.currrec)

{

temp.currindex = temp.HashFunc(tempkey);

int ind = temp.currindex;

if (temp.linerec[temp.currindex]->key == tempkey)

return temp.linerec[temp.currindex]->data;

else

{

while (temp.flag[temp.currindex] && ((temp.currindex + 1) != ind) && (temp.linerec[temp.currindex]->key != tempkey))

temp.currindex = (temp.currindex + 1) % temp.maxrec;

if (temp.linerec[temp.currindex]->key == tempkey)

return temp.linerec[temp.currindex]->data;

else

throw "key is not found";

}

}

else

throw "table is empty";

}

template<typename val>

void HashTab<val>::insert(const string& tempkey, const val& tempdata)

{

if (IsFull())

Realloc();

currindex = HashFunc(tempkey);

if (!flag[currindex])

{

linerec[currindex] = new TabRec<val>(tempkey, tempdata);

currrec++;

flag[currindex] = 1;

}

else

if (linerec[currindex]->key != tempkey)

{

int ind = currindex;

while (flag[currindex] && ((currindex + 1) != ind))

currindex = (currindex + 1) % maxrec;

linerec[currindex] = new TabRec<val>(tempkey, tempdata);

currrec++;

flag[currindex] = 1;

}

else

throw "key already exists";

}

template<typename val>

void HashTab<val>::del(const string& key)

{

reset();

if (currrec)

{

currindex = HashFunc(key);

int ind = currindex;

if (flag[currindex])

{

if (linerec[currindex]->key != key)

{

while (flag[currindex] && ((currindex + 1) != ind) && (linerec[currindex]->key != key))

currindex = (currindex + 1) % maxrec;

if (!flag[currindex] || linerec[currindex]->key != key)

throw "key is not found";

else

{

linerec[currindex] = new TabRec<val>;

flag[currindex] = 0;

currrec--;

}

}

else

{

linerec[currindex] = new TabRec<val>;

flag[currindex] = 0;

currrec--;

}

}

else

throw "key is not found";

}

else

throw "table is empty";

}

template<typename val>

val& HashTab<val>::getcurr() const

{

if (currrec)

return linerec[currindex]->data;

else

throw "table is empty";

}

template<typename val>

void HashTab<val>::set()

{

if (currrec)

{

currindex++;

while (flag[currindex] != 1)

currindex = (currindex + 1) % maxrec;

}

else

throw "table is empty";

}

template <typename val>

ostream& operator<< (ostream& os, const HashTab<val>& t)

{

if (t.currrec)

{

for (int i = 0; i < t.maxrec; i++)

if (t.flag[i])

os << i << " | " << t.linerec[i]->key << " | " << t.linerec[i]->data << endl;

}

else

os << "Table is empty" << endl;

return os;

}

##### Тесты для ScanTab

#include "scan\_table.h"

#include <gtest.h>

TEST(ScanTab, can\_create\_empty)

{

ASSERT\_NO\_THROW(ScanTab<int> t);

}

TEST(ScanTab, can\_copied\_from\_empty)

{

ScanTab<int> t;

ASSERT\_NO\_THROW(ScanTab<int> tt(t));

}

TEST(ScanTab, can\_create\_not\_empty)

{

ASSERT\_NO\_THROW(ScanTab<int> t(2));

}

TEST(ScanTab, can\_copied\_from\_not\_empty)

{

ScanTab<int> t(2);

ASSERT\_NO\_THROW(ScanTab<int> tt(t));

}

TEST(ScanTab, can\_realloc\_memory\_when\_is\_necessary)

{

ScanTab<int> t;

ASSERT\_NO\_THROW(t.insert("1", 2));

}

TEST(ScanTab, can\_reset)

{

ScanTab<int> t;

t.insert("1", 1);

t.reset();

EXPECT\_EQ(t.getcurrind(), 0);

}

TEST(ScanTab, can\_setnext)

{

ScanTab<int> t(4);

t.insert("1", 1);

t.insert("2", 1);

t.reset();

t.set();

EXPECT\_EQ(t.getcurrind(), 1);

}

TEST(ScanTab, can\_get\_current\_data)

{

ScanTab<int> t;

t.insert("1", 1);

t.reset();

t.set();

EXPECT\_EQ(t.getcurr(), 1);

}

TEST(ScanTab, can\_insert)

{

ScanTab<int> t;

ASSERT\_NO\_THROW(t.insert("1", 1));

}

TEST(ScanTab, throw\_when\_getcurr\_with\_no\_records)

{

ScanTab<int> t;

ASSERT\_ANY\_THROW(t.getcurr());

}

TEST(ScanTab, throw\_when\_set\_with\_no\_records)

{

ScanTab<int> t;

ASSERT\_ANY\_THROW(t.set());

}

TEST(ScanTab, cannot\_search\_when\_table\_is\_empty)

{

ScanTab<int> t;

ASSERT\_ANY\_THROW(t.search("1"));

}

TEST(ScanTab, can\_search)

{

ScanTab<int> t;

t.insert("1", 1);

EXPECT\_EQ(t.search("1"), 1);

}

TEST(ScanTab, can\_search2)

{

ScanTab<int> t;

t.insert("2", 1);

ASSERT\_ANY\_THROW(t.search("1"));

}

TEST(ScanTab, can\_del1)

{

ScanTab<int> t;

t.insert("1", 1);

t.del("1");

EXPECT\_EQ(t.IsEmpty(), true);

}

TEST(ScanTab, can\_del2)

{

ScanTab<int> t(4);

//t.insert("3", 5);

t.insert("1", 1);

t.insert("2", 2);

t.del("1");

t.del("2");

EXPECT\_EQ(t.IsEmpty(), true);

}

TEST(ScanTab, cannot\_del\_from\_empty\_table)

{

ScanTab<int> t;

ASSERT\_ANY\_THROW(t.del("1"));

}

##### Тесты для SortTab

#include "sort\_table.h"

#include <gtest.h>

TEST(SortTab, can\_create\_empty)

{

ASSERT\_NO\_THROW(SortTab<int> t);

}

TEST(SortTab, can\_copied\_from\_empty)

{

SortTab<int> t;

ASSERT\_NO\_THROW(SortTab<int> tt(t));

}

TEST(SortTab, can\_create\_not\_empty)

{

ASSERT\_NO\_THROW(SortTab<int> t(2));

}

TEST(SortTab, can\_copied\_from\_not\_empty)

{

SortTab<int> t(2);

ASSERT\_NO\_THROW(SortTab<int> tt(t));

}

TEST(SortTab, can\_realloc\_memory\_when\_is\_necessary)

{

SortTab<int> t;

ASSERT\_NO\_THROW(t.insert("1", 2));

}

TEST(SortTab, can\_reset)

{

SortTab<int> t;

t.insert("1", 1);

t.reset();

EXPECT\_EQ(t.getcurrind(), 0);

}

TEST(SortTab, can\_setnext)

{

SortTab<int> t;

t.insert("1", 1);

t.insert("2", 1);

t.reset();

t.set();

EXPECT\_EQ(t.getcurrind(), 1);

}

TEST(SortTab, can\_get\_current\_data)

{

SortTab<int> t;

t.insert("1", 1);

t.reset();

t.set();

EXPECT\_EQ(t.getcurr(), 1);

}

TEST(SortTab, can\_insert)

{

SortTab<int> t;

ASSERT\_NO\_THROW(t.insert("1", 1));

}

TEST(SortTab, throw\_when\_getcurr\_with\_no\_records)

{

SortTab<int> t;

ASSERT\_ANY\_THROW(t.getcurr());

}

TEST(SortTab, throw\_when\_set\_with\_no\_records)

{

SortTab<int> t;

ASSERT\_ANY\_THROW(t.set());

}

TEST(SortTab, cannot\_search\_when\_table\_is\_empty)

{

SortTab<int> t;

ASSERT\_ANY\_THROW(t.search("1"));

}

TEST(SortTab, can\_search)

{

SortTab<int> t;

t.insert("1", 1);

EXPECT\_EQ(t.search("1"), 1);

}

TEST(SortTab, can\_del)

{

SortTab<int> t;

t.insert("1", 1);

t.del("1");

EXPECT\_EQ(t.IsEmpty(), true);

}

TEST(SortTab, can\_del2)

{

SortTab<int> t;

t.insert("1", 1);

t.insert("2", 2);

t.del("1");

t.del("2");

EXPECT\_EQ(t.IsEmpty(), true);

}

TEST(SortTab, cannot\_del\_from\_empty\_table)

{

SortTab<int> t;

ASSERT\_ANY\_THROW(t.del("1"));

}

##### Tecты для HashTab

#include "hash\_table.h"

#include "polinom.h"

#include <gtest.h>

TEST(HashTab, can\_create\_empty)

{

ASSERT\_NO\_THROW(HashTab<int> t);

}

TEST(HashTab, can\_copied\_from\_empty)

{

HashTab<int> t;

ASSERT\_NO\_THROW(HashTab<int> tt(t));

}

TEST(HashTab, can\_create\_not\_empty)

{

ASSERT\_NO\_THROW(HashTab<int> t(2));

}

TEST(HashTab, can\_copied\_from\_not\_empty)

{

HashTab<int> t(2);

ASSERT\_NO\_THROW(HashTab<int> tt(t));

}

TEST(HashTab, can\_realloc\_memory\_when\_is\_necessary)

{

HashTab<int> t;

ASSERT\_NO\_THROW(t.insert("1", 2));

}

TEST(HashTab, can\_reset1)

{

HashTab<int> t;

t.insert("1", 1);

t.insert("2", 1);

t.reset();

EXPECT\_EQ(t.getcurrind(), 0);

}

TEST(HashTab, can\_reset2)

{

HashTab<int> t;

t.reset();

EXPECT\_EQ(t.getcurrind(), -1);

}

TEST(HashTab, can\_setnext)

{

HashTab<int> t(2);

t.insert("1", 1);

t.set();

EXPECT\_EQ(t.getcurrind(), 1);

}

TEST(HashTab, cant\_setnext\_in\_empty\_table)

{

HashTab<int> t;

ASSERT\_ANY\_THROW(t.set());

}

TEST(HashTab, can\_get\_current\_data)

{

HashTab<int> t;

t.insert("1", 1);

t.reset();

t.set();

EXPECT\_EQ(t.getcurr(), 1);

}

TEST(HashTab, can\_insert)

{

HashTab<int> t;

ASSERT\_NO\_THROW(t.insert("1", 1));

}

TEST(HashTab, throw\_when\_getcurr\_with\_no\_records)

{

HashTab<int> t;

ASSERT\_ANY\_THROW(t.getcurr());

}

TEST(HashTab, throw\_when\_set\_with\_no\_records)

{

HashTab<int> t;

ASSERT\_ANY\_THROW(t.set());

}

TEST(HashTab, cannot\_search\_when\_table\_is\_empty)

{

HashTab<int> t;

ASSERT\_ANY\_THROW(t.search("1"));

}

TEST(HashTab, can\_search)

{

HashTab<int> t;

t.insert("1", 1);

EXPECT\_EQ(t.search("1"), 1);

}

TEST(HashTab, can\_del1)

{

HashTab<int> t;

t.insert("1", 1);

t.del("1");

EXPECT\_EQ(t.IsEmpty(), true);

}

TEST(HashTab, can\_del2)

{

HashTab<int> t;

t.insert("1", 1);

t.insert("2", 2);

t.del("1");

t.del("2");

EXPECT\_EQ(t.IsEmpty(), true);

}

TEST(HashTab, cannot\_del\_from\_empty\_table)

{

HashTab<int> t;

ASSERT\_ANY\_THROW(t.del("1"));

}